Basics

In [1]:
```
print("Hello, World!")
```

```
Hello, World!
```

In [2]:
```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

```
Five is greater than two!
Five is greater than two!
```

In [3]:
```
#This is a comment
print("Hello, World!")
```

```
Hello, World!
```

In [4]:
```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

```
Hello, World!
```

Python Casting

In [5]:
```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

Python Strings

In [6]:
```
a = "Hello, World!"
print(a[1])
```

```
e
```

In [7]:
```
for x in "banana":
  print(x)
```

```
b
a
n
a
n
a
```

In [8]:
```
a = "Hello, World!"
print(len(a))
```

```
13
```

In [9]:
```
txt = "The best things in life are free!"
print("free" in txt)
```

```
True
```

In [10]:
```
txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")
```

```
Yes, 'free' is present.
```

In [11]:
```
txt = "The best things in life are free!"
if "expensive" not in txt:
  print("No, 'expensive' is NOT present.")
```

```
No, 'expensive' is NOT present.
```

Slicing

In [12]:
```
#Get the characters from position 2 to position 5 (not included):

b = "Hello, World!"
print(b[2:5])
```

```
llo
```

In [14]:
```
#Get the characters from the start to position 5 (not included):

b = "Hello, World!"
print(b[:5])
```

```
Hello
```

In [15]:
```
b = "Hello, World!"
print(b[-5:-2])
```

```
orl
```

Python - Modify Strings

In [17]:
```
a = "Hello, World!"
print(a.upper())
a = "Hello, World!"
print(a.lower())
```

```
HELLO, WORLD!
hello, world!
```

In [18]:
```
#The strip() method removes any whitespace from the beginning or the end:
```

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

```
Hello, World!
```

In [19]:

```
#The replace() method replaces a string with another string:

a = "Hello, World!"
print(a.replace("H", "J"))
```

```
Jello, World!
```

In [21]:

```
#The split() method splits the string into substrings if it finds instances of the s

a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

```
['Hello', ' World!']
```

In [22]:

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

```
Hello World
```

In [23]:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

```
My name is John, and I am 36
```

In [24]:

```
txt = "We are the so-called \"Vikings\" from the north."
print(txt)
```

```
We are the so-called "Vikings" from the north.
```

String Methods

In [25]:

```
#Return the number of times the value "apple" appears in the string:
txt = "I love apples, apple are my favorite fruit"

x = txt.count("apple")

print(x)
```

```
2
```

In [26]:

```
txt = "My name is Ståle"

x = txt.encode()

print(x)
```

```
b'My name is St\xc3\xa5le'
```

In [27]:

```
#Check if the string ends with a punctuation sign (.):

txt = "Hello, welcome to my world."
```

```
x = txt.endswith(".")

print(x)
```

```
True
```

In [28]:

```
#Where in the text is the word "welcome"?:

txt = "Hello, welcome to my world."

x = txt.index("welcome")

print(x)
```

```
7
```

In [29]:

```
#string.index(value, start, end)
txt = "Hello, welcome to my world."

x = txt.index("e", 5, 10)

print(x)
```

```
8
```

In [30]:

```
txt = "I like bananas"

x = txt.replace("bananas", "apples")

print(x)
```

```
I like apples
```

In [31]:

```
txt = "apple#banana#cherry#orange"

x = txt.split("#")

print(x)
```

```
['apple', 'banana', 'cherry', 'orange']
```

In [32]:

```
txt = "apple#banana#cherry#orange"

# setting the maxsplit parameter to 1, will return a list with 2 elements!
x = txt.split("#", 1)

print(x)
```

```
['apple', 'banana#cherry#orange']
```

In [33]:

```
x = 4
y = x**3
print(y)
```

```
64
```

Python Lists

In [34]:
```
mylist = ["apple", "banana", "cherry"]
```

In [36]:
```
thislist = ["apple", "banana", "cherry"]
print(thislist)
print(len(thislist))
```

```
['apple', 'banana', 'cherry']
3
```

In [37]:
```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
banana
```

In [38]:
```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

```
cherry
```

In [39]:
```
#Return the third, fourth, and fifth item:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

```
['cherry', 'orange', 'kiwi']
```

In [40]:
```
#This example returns the items from the beginning to, but NOT including, "kiwi":

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

```
['apple', 'banana', 'cherry', 'orange']
```

In [41]:
```
#This example returns the items from "orange" (-4) to, but NOT including "mango" (-1

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

```
['orange', 'kiwi', 'melon']
```

In [42]:
```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

```
Yes, 'apple' is in the fruits list
```

In [43]:
```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

```
['apple', 'blackcurrant', 'cherry']
```

In [44]:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

```
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
```

Insert Items

In [45]:
```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

```
['apple', 'banana', 'watermelon', 'cherry']
```

Append Items

In [46]:
```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

Extend List

In [47]:
```
#Add the elements of tropical to thislist:

thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

```
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']
```

Remove Specified Item

In [48]:
```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

```
['apple', 'cherry']
```

In [49]:
```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

```
['apple', 'cherry']
```

In [50]:
```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
['banana', 'cherry']
```

In [51]:
```
thislist = ["apple", "banana", "cherry"]
del thislist
```

In [52]:
```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

```
[]
```

Python - Loop Lists

In [53]:
```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

```
apple
banana
cherry
```

In [54]:
```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

```
apple
banana
cherry
```

In [55]:
```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

```
apple
banana
cherry
```

In [56]:
```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

```
apple
banana
cherry
```

Out[56]:
```
[None, None, None]
```

Python - List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

In [57]:
```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

```
['apple', 'banana', 'mango']
```

In [58]:
```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

```
['apple', 'banana', 'mango']
```

In [59]:
```
newlist = [x for x in fruits if x != "apple"]
```

Python - Sort Lists

In [60]:
```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

```
['banana', 'kiwi', 'mango', 'orange', 'pineapple']
```

In [61]:
```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

```
[23, 50, 65, 82, 100]
```

In [62]:
```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

```
['pineapple', 'orange', 'mango', 'kiwi', 'banana']
```

In [63]:
```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)
```

```
[100, 82, 65, 50, 23]
```

Reverse Order

In [64]:
```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

```
['cherry', 'Kiwi', 'Orange', 'banana']
```

Python - Copy Lists

In [65]:
```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

```
['apple', 'banana', 'cherry']
```

In [66]:
```
#Join two list:

list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

```
['a', 'b', 'c', 1, 2, 3]
```

In [67]:
```
#Use the extend() method to add list2 at the end of list1:

list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

Python Tuples

In [68]:
```
mytuple = ("apple", "banana", "cherry")
```

In [69]:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

In [70]:
```
print(len(thistuple))
```

```
3
```

In [71]:
```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

```
<class 'tuple'>
<class 'str'>
```

In [72]:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

In [73]:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
cherry
```

In [74]:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

```
('cherry', 'orange', 'kiwi')
```

In [75]:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

```
('apple', 'banana', 'cherry', 'orange')
```

In [76]:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

```
('cherry', 'orange', 'kiwi', 'melon', 'mango')
```

Python - Update Tuples

In [77]:
```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

```
('apple', 'kiwi', 'cherry')
```

Add Items

In [78]:
```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```

In [79]:
```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

```
('apple', 'banana', 'cherry', 'orange')
```

Remove Items

In [80]:
```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

In [82]:
```
thistuple = ("apple", "banana", "cherry")
del thistuple
# print(thistuple) #this will raise an error because the tuple no longer exists
```

Unpacking a Tuple

In [83]:
```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

```
apple
banana
cherry
```

Using Asterisk*

In [84]:
```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

```
apple
banana
['cherry', 'strawberry', 'raspberry']
```

In [85]:
```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)
```

```
apple
['mango', 'papaya', 'pineapple']
cherry
```

Join Tuples

In [86]:
```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

In [87]:
```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

In [88]:
```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.count(5)

print(x)
```

```
2
```

In [89]:
```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.index(8)

print(x)
```

```
3
```

Python Sets

In [90]:
```
myset = {"apple", "banana", "cherry"}
```

In [91]:
```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

```
{'banana', 'apple', 'cherry'}
```

In [92]:
```
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

```
{'banana', 'apple', 'cherry'}
```

In [93]:
```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

```
{'banana', 'apple', 'cherry'}
```

In [94]:
```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

```
banana
apple
cherry
```

Add Item into SET

In [95]:
```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

```
{'banana', 'apple', 'cherry', 'orange'}
```

In [96]:
```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)
```

```
{'banana', 'cherry', 'orange', 'kiwi', 'apple'}
```

Remove Item

In [97]:
```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

```
{'apple', 'cherry'}
```

In [98]:
```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

```
{'apple', 'cherry'}
```

In [99]:
```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

```
banana
{'apple', 'cherry'}
```

In [100…
```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

```
set()
```

In [102…
```
thisset = {"apple", "banana", "cherry"}

del thisset

#print(thisset)
```

In [102…
```
thisset = {"apple", "banana", "cherry"}

del thisset

#print(thisset)
```

Set Operations

In [103…
```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
{1, 2, 3, 'a', 'c', 'b'}
```

In [104…
```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

```
{1, 2, 3, 'a', 'c', 'b'}
```

In [105…
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)

print(x)
```

```
{'apple'}
```

In [106…
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.symmetric_difference_update(y)

print(x)
```

```
{'banana', 'microsoft', 'cherry', 'google'}
```

Python Dictionaries

In [107…
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

In [108…
```
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

In [109…
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

```
Ford
```

In [110…
```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

In [111…
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(type(thisdict))
```

```
<class 'dict'>
```

In [112…

```
thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)
```

```
{'name': 'John', 'age': 36, 'country': 'Norway'}
```

In [1]:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```

```
dict_keys(['brand', 'model', 'year'])
dict_keys(['brand', 'model', 'year', 'color'])
```

In [2]:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

```
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 2020])
```

In [3]:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["color"] = "red"

print(x) #after the change
```

```
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 1964, 'red'])
```

Removing Items

In [5]:

```
thisdict = {
  "brand": "Ford",
```

```
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

In [6]:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

In [7]:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

In [8]:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

In [9]:

```
#Increment the sequence with 3 (default is 1):

for x in range(2, 30, 3):
  print(x)
```

```
2
5
8
11
14
17
20
23
26
29
```

Arbitrary Arguments, *args

In [10]:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
The youngest child is Linus
```

In [11]:
```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

```
His last name is Refsnes
```

Python Lambda

In [12]:
```
x = lambda a : a + 10
print(x(5))
```

```
15
```

In [13]:
```
x = lambda a, b : a * b
print(x(5, 6))
```

```
30
```

In [14]:
```
def myfunc(n):
  return lambda a : a * n
```

Why Use Lambda Functions? The power of lambda is better shown when you use them as an anonymous function inside another function.

In [15]:
```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

```
22
33
```

Exception Handling

In [16]:
```
try:
  print(x)
except:
  print("An exception occurred")
```

```
<function <lambda> at 0x00000141A76399D0>
```

In [17]:
```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

```
<function <lambda> at 0x00000141A76399D0>
```

In [18]:
```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

```
<function <lambda> at 0x00000141A76399D0>
The 'try except' is finished
```

In [19]:
```
username = input("Enter username:")
print("Username is: " + username)
```

```
Username is: hfyu
```

File Handling

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In [ ]:
```
f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

In [ ]:
```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

In [ ]:
```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

In [ ]:
```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

map, reduce and filter functions

In [22]:
```
def multiplyNumbers(givenNumbers):
   return givenNumbers*3
givenNumbers = map(multiplyNumbers, [1, 3, 5, 2, 6, 10])

print(list(givenNumbers))
```

```
[3, 9, 15, 6, 18, 30]
```

In [23]:

```
def votingAge(givenNumumber):
   if givenNumumber>=18:
      return givenNumumber
inputList = [3, 20, 18, 6, 14, 25, 19]
result_filterObj = filter(votingAge, inputList)
print(list(result_filterObj))
```

```
[20, 18, 25, 19]
```

In [25]:

```
from functools import reduce
def addNumbers(x, y):
   return x+y
inputList = [12, 4, 10, 15, 6, 5]
print("The sum of all list items:")
print(reduce(addNumbers, inputList))
```

```
The sum of all list items:
52
```

Mutability vs Immutability

In programming, you have an immutable object if you can't change the object's state after you've created it. In contrast, a mutable object allows you to modify its internal state after creation. In short, whether you're able to change an object's state or contained data is what defines if that object is mutable or immutable.

Immutable Built-in Data Types in Python Numbers Booleans Strings Bytes Tuples Mutable Built-in Data Types in Python Lists Dictionaries Sets

Questions And Answers

1 What is the difference between a Set and Dictionary?

=> The set is an unordered collection of data types that is iterable, mutable and has no duplicate elements. A dictionary in Python is an ordered collection of data values, used to store data values like a map.

2 What is List Comprehension? Give an Example.

=> List comprehension is a syntax construction to ease the creation of a list based on existing iterable.

For Example:

my_list = [i for i in range(1, 10)]

3 What is a lambda function?

=> A lambda function is an anonymous function. This function can have any number of parameters but, can have just one statement. For Example:

a = lambda x, y : x*y print(a(7, 19))

4. What are *args and *kwargs?

=> To pass a variable number of arguments to a function in Python, use the special syntax *args and **kwargs in the function specification.

5. What is a zip function?

   => Python zip() function returns a zip object, which maps a similar index of multiple containers. It takes an iterable, converts it into an iterator and aggregates the elements based on iterables passed. It returns an iterator of tuples.

In [ ]: